Advances in the NetCDF Data Model, Format, and Software

Russ Rew
Coauthors: John Caron, Ed Hartnett, Dennis Heimbigner

UCAR Unidata
December 2010

UCAR COMMUNITY PROGRAMS

Outline

- Background
- Recent advances
  - Refactoring for interoperability
  - Performance improvements
  - Experience adapting software to enhanced data model
  - Standards status
- State of adoption of netCDF-4
- Summary

NetCDF: more than a format

- Data model
  - netCDF-3 classic data model: Variables, Dimensions, Attributes
  - netCDF-4 enhanced data model: adds Groups, user-defined Types
- File format
  - classic format, 64-bit variant
  - netCDF-4 (HDF5-based) format, classic model variant
- Application programming interfaces (APIs)
  - C-based APIS: C, Fortran, C++, Python, Perl, Ruby, MATLAB, IDL, …
  - Java API: Java, MATLAB

Together, the data model, file format, and APIs support the creation, access, and sharing of scientific data

What is netCDF?

Development Milestones

1989: portable, self-describing data format, data model, and software for creation, access, and sharing of scientific data

1990's: growth of use in ocean and climate models, 3rd-party software support (NCO, NCL, IDL, MATLAB)

2002: Java version with OPeNDAP client support

2003: NASA funded netCDF-4/HDF5 project; Argonne/ Northwestern parallel netCDF

2004: netCDF-Java plug ins for reading other formats, NcML aggregation service

2007: netCDF-Java Common Data Model (access to other formats through netCDF interface)

2008: netCDF-4 C and Fortran library with HDF5 integration, enhanced data model, parallel I/O

2009: “netCDF classic” format standard endorsed

2010: version 4.1.1 - OPeNDAP client support for C/ Fortran libraries; udunits, CF library support; pnetcdf, HDF4 access

The netCDF “classic” data model, in UML

NetCDF Data has

- **Variables** (eg *temperature, pressure)*
- **Attributes** (eg *units)*
- **Dimensions** (eg *lat, lon, level, time)*

Variables have

- Name, shape, type, attributes
- N-dimensional array of values

Dimensions have

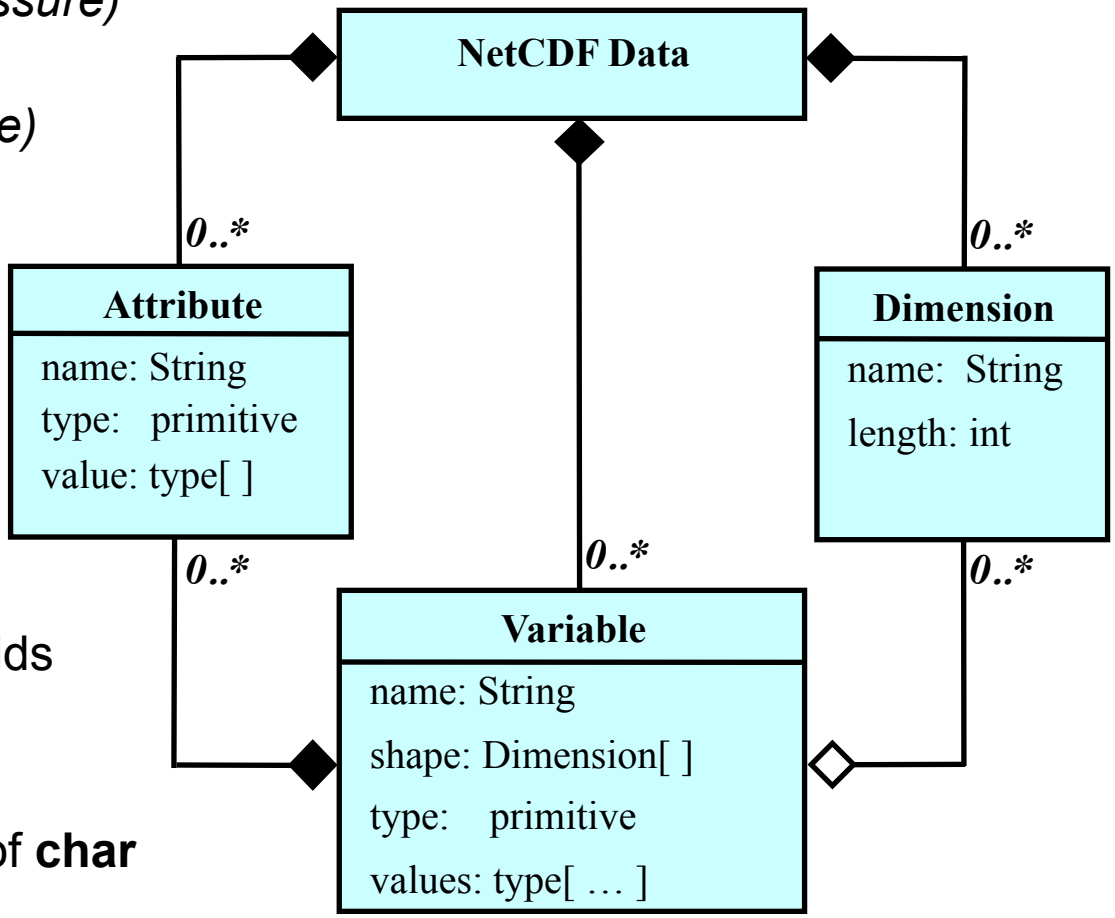
- Name, length
- One dimension may grow

Variables *may share* dimensions

- Represents shared coordinates, grids

Six Primitive types

- 8-bit **byte**, 16-bit **short**, 32-bit **int**,
- 32-bit **float**, 64-bit **double,** arrays of **char**



NetCDF classic data model

Strengths

- ✓ Data model simple to understand and explain
- ✓ Efficient implementation freely available
- ✓ Generic applications easy to develop
- ✓ Representation good for gridded multidimensional data
- ✓ Shared dimensions useful for coordinate systems

Limitations

- Small set of primitive types
- Flat data model limited to multidimensional arrays, lists, (name, value) pairs
- Flat name space not ideal for organizing data objects
- Lacks nested structures, variable-length types, enumerations

NetCDF classic format

Strengths

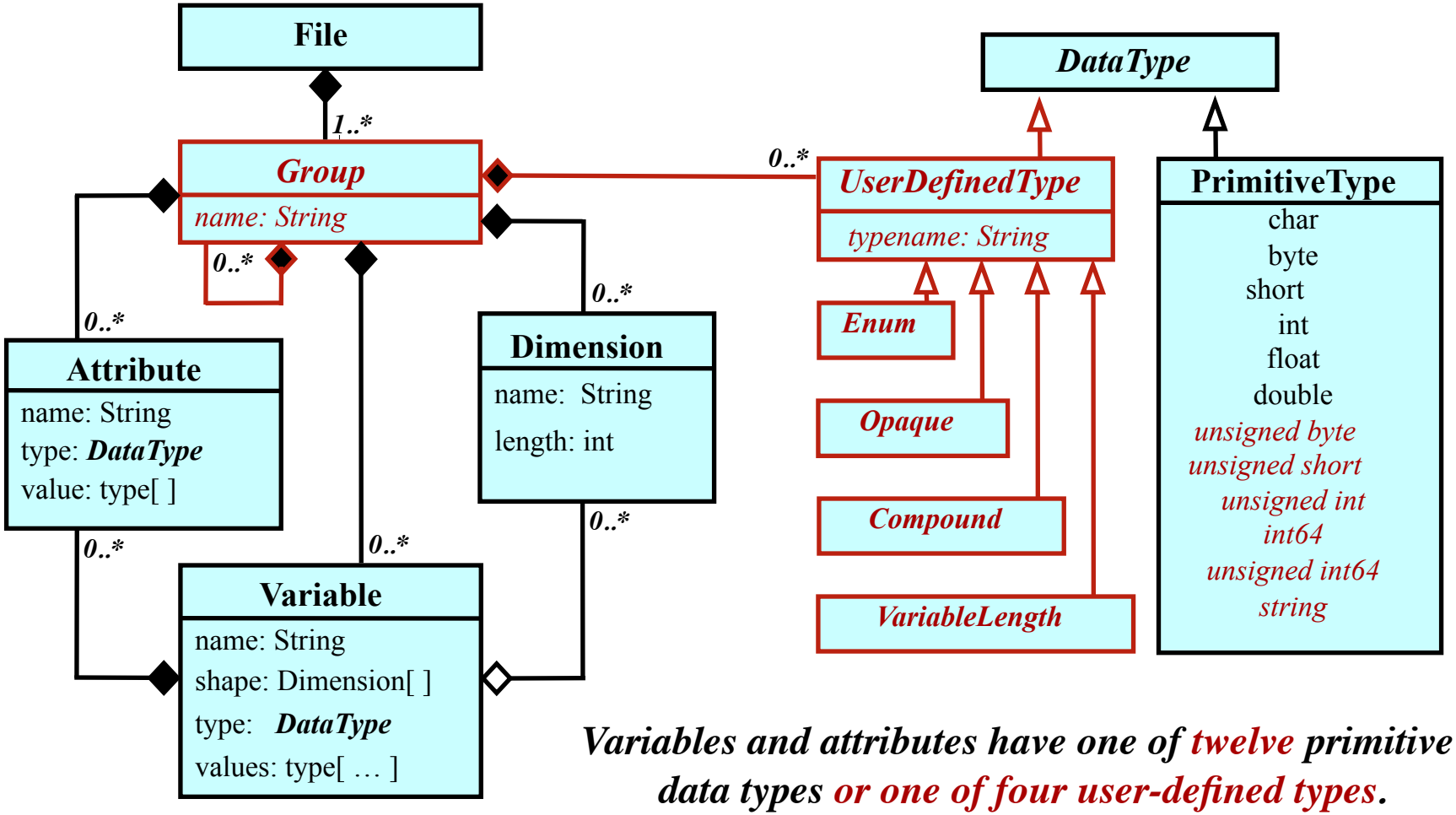
- ✓ Simple to understand and explain
- ✓ Supported by many applications
- ✓ Standard used in many archives, data projects
- ✓ Mature conventions and best practices have evolved

Limitations

- Schema changes may be costly
- No support for compression
- Only one dimension can grow efficiently
- Portable representation favors big-endian platforms

The netCDF-4 *enhanced* data model

A file has a top-level unnamed group. Each group may contain one or more named subgroups, user-defined types, variables, dimensions, and attributes. Variables also have attributes. Variables may share dimensions, indicating a common grid. One or more dimensions may be of unlimited length.



Variables and attributes have one of twelve primitive data types or one of four user-defined types.

NetCDF enhanced data model

Strengths

- ✓ Simpler than HDF5, with similar representational power
- ✓ Adds shared dimensions to HDF5 data model
- ✓ Continues support for existing data, software, and conventions
- ✓ Eliminates netCDF classic model limitations
- ✓ Provides nested structures: hierarchical groups, recursive data types
- ✓ Independent features permit incremental adaptation, adoption

On the other hand

- More complex than classic data model
- More effort required to develop general tools and applications
- Not yet widely adopted
- Hence, no comprehensive best practices and conventions yet

(Data Model, Format) combinations

- (Classic, Classic)
  - Mature conventions, best practices (e.g. CF Conventions)
  - Maximum portability, compatibility with old software
- (Classic, netCDF-4)
  - Requires only relinking instead of modifying software
  - Performance benefits: compression, chunking, larger variables, efficient schema changes
- (Enhanced, netCDF-4)
  - Additional data types, including user-defined
  - Advantages in modeling data, including observational data
  - High Performance Computing applications
  - Datasets with large number of data objects
  - Reading other kinds of data (HDF4, HDF5, relational, …)

Recent advances

Standards

Refactoring architecture for interoperability

Performance improvements

Generic tools

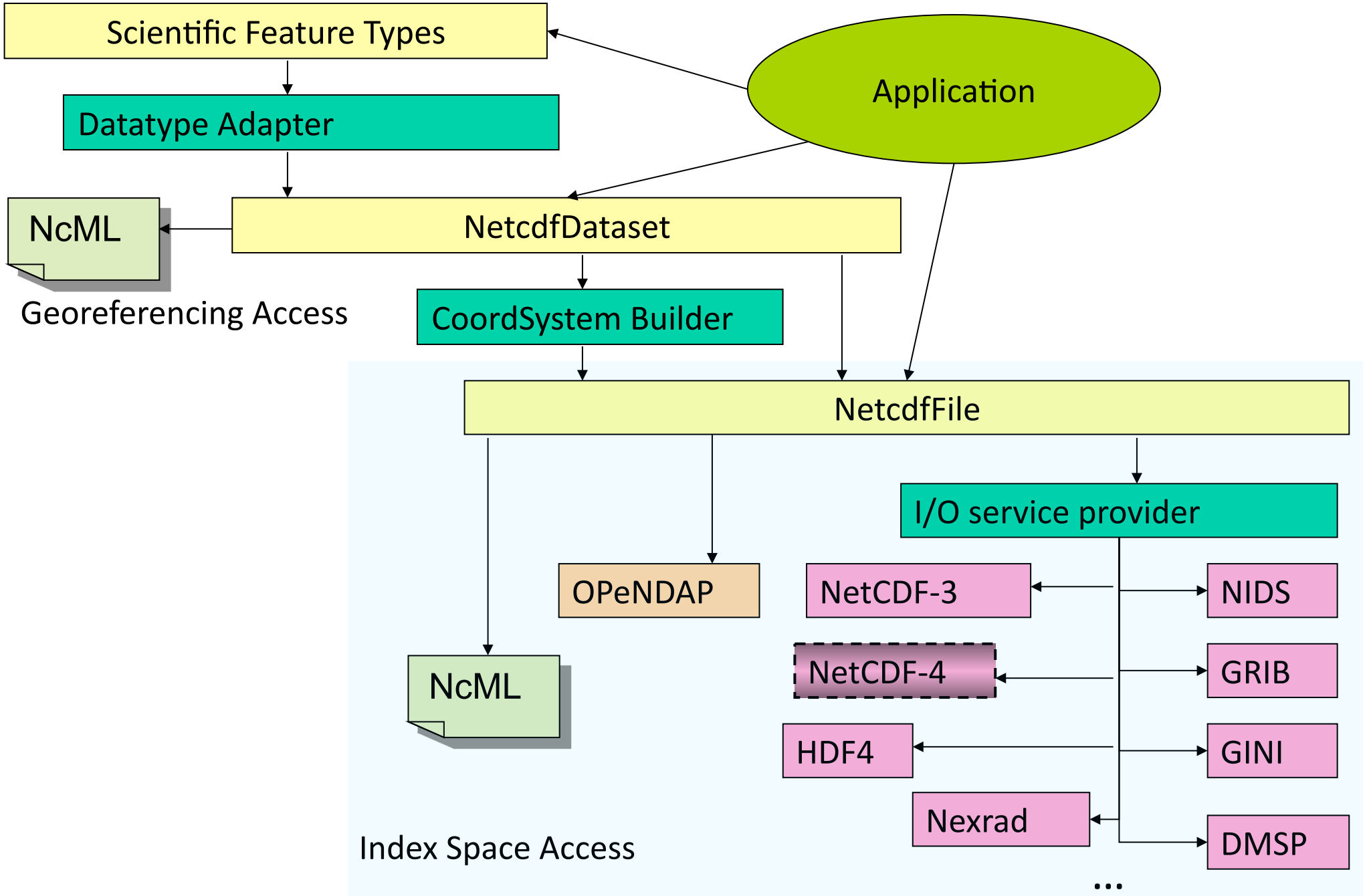
Practical experience

Status of netCDF-4 adoption

Standards: from traction to sanction

- *2009-02-05:* NASA Earth Science Data Systems (ESDS) Standards Process Group endorsed **netCDF classic and 64-bit offset formats** as appropriate for NASA Earth Science data.
- *2010-03-1:* Integrated Ocean Observing System (IOOS) Data Management and Communications (DMAC) Subsystem endorsed **netCDF with Climate and Forecast (CF) conventions** as a preferred data format.
- *2010-09-27:* Steering Committee of the Federal Geographic Data Committee (FGDC) officially endorsed **netCDF** as a Common Encoding Standard.
- *2010-11-05:* Open Geospatial Consortium (OGC) began vote on approving "OGC Network Common Data Form (**NetCDF**) Core Encoding Standard version 1.0 ” as a new OGC standard. The vote closes on January 4, 2011.

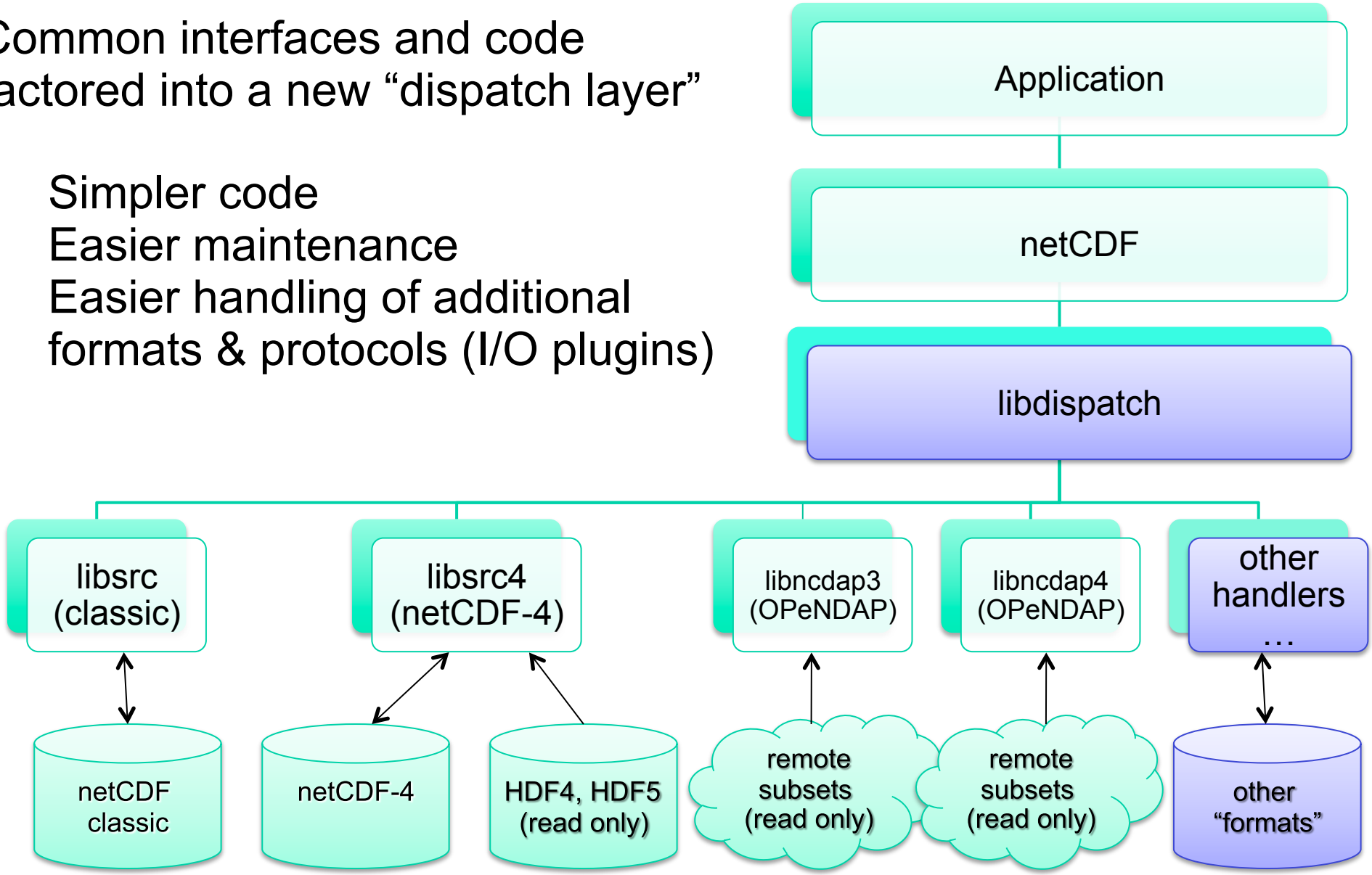
NetCDF-Java/Common Data Model architecture



C library refactored for interoperability

Common interfaces and code factored into a new “dispatch layer”

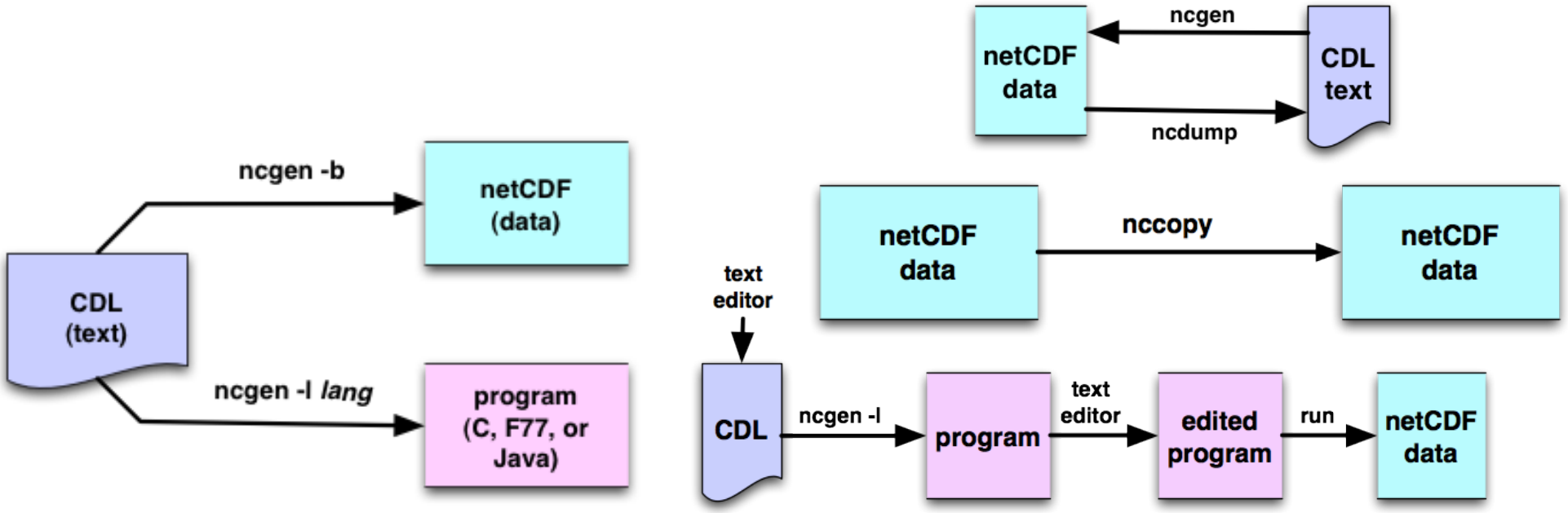
- Simpler code
- Easier maintenance
- Easier handling of additional formats & protocols (I/O plugins)



Performance improvements

- Refactored read code for large speedup on opening netCDF-4 files with compressed or chunked variables
- Speedup variable and dimension lookup by name
- Improved memory allocation to reduce memory footprint
- Reduced memory when parallel I/O used
- Eliminated memory leaks
- Improved read code w.r.t. handling a large number of netCDF-4 attributes and variables
- Applied intelligent caching to remote access for OPeNDAP client
- Some of these improvements are in upcoming version 4.1.2

Generic tools



- Adapted generic tools to netCDF-4 enhanced data model

  ncdump: converts netCDF data to CDL text form
  ncgen: converts CDL text to netCDF data or generates program
  nccopy: copies netCDF data, optionally converting to a different form
- Proved practicality of handling potentially infinite number of user-defined nested
- Tool adaptation led to API additions

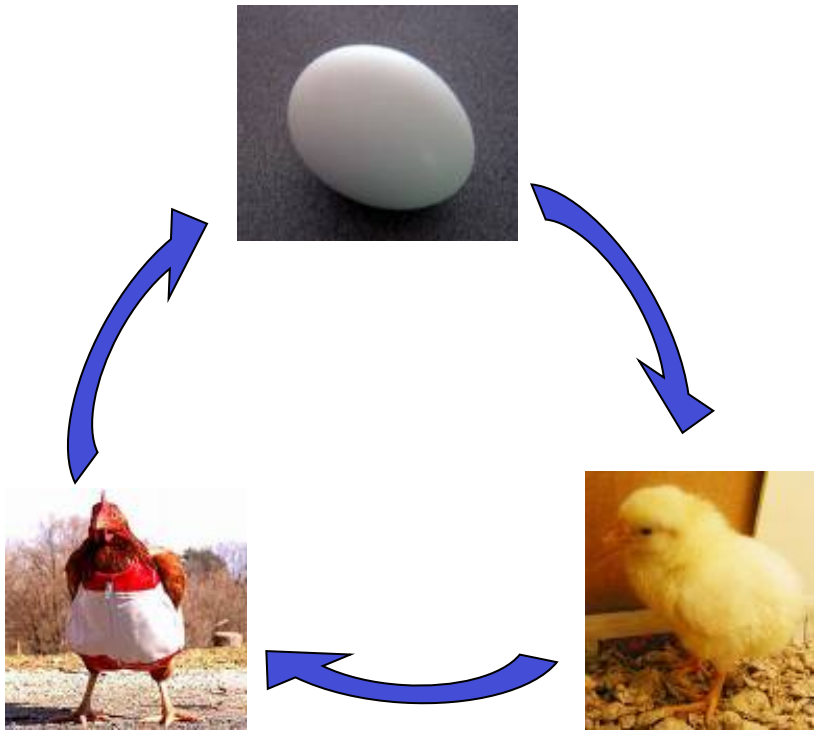
Experience developing nccopy utility

- Shows any netCDF-4 data can be accessed through API without previous or built-in knowledge of user-defined data types
- Showed netCDF-4 API is adequate for handling arbitrary nesting of groups and user-defined types
- Provides evidence that programming generic netCDF-4 applications is not too difficult
  - Classic data model: 500 lines of C
  - Enhanced data model: 900 lines of C
- Demonstrates usefulness of additional higher-level APIs for tool developers
  - Iterator APIs for simpler data access
  - APIs that make recursion unnecessary (e.g. visiting groups, comparing values of a user-defined type)

Practical experience

- Most experience to date is with netCDF-4 classic model format
  - uses netCDF-3 classic data model, APIs
  - uses netCDF-4 HDF5-based format
  - provides backward compatibility
  - Enables performance features: compression, multidimensional tiling (chunking), efficient schema changes, parallel I/O, …
- Adoption proceeding smoothly in a 3-step process
  1. Relink applications with netCDF-4 library
  2. Continue use of classic model, netCDF-3 APIs but with netCDF-4 classic model format to get performance benefits
  3. Make use of features of enhanced model, as needed/supported

Last year: game of “chicken”; who goes first?



- Data producers
  - Waiting until netCDF enhanced data model features are supported by more software, development of conventions
- Developers
  - Waiting for netCDF data that requires enhanced model and for development of conventions
- Convention creators
  - Waiting for data providers and software developers to identify needs for new conventions based on usage experience
- Result: “chicken-and-egg logjam”
  - *Delays effective use of advances in scientific data models for large and complex collections*

Status of netCDF-4 adoption: Logjam appears to be broken

- **NetCDF-4 enhanced model** support in language APIs: C, Java (read only), C++ (beta), Fortran
- Partial support for netCDF-4 enhanced model also in NCO, NCL, Panoply, Python API, …
- **NetCDF-4 classic model** support in analysis and visualization apps: IDL, GrADS, CDAT, MATLAB, IDV, NCO, NCL, CDO, PyNGL, ncview, Panoply, Ferret, OGC WMS and WCS clients
- Data providers using **netCDF-4 classic model format** for transparent compression and chunking: groups in NASA, NOAA, GFDL, COLA
- CMIP5 decided to continue using **classic model and classic format** (no compression) due to time accessing compressed data on server

Concluding Remarks

- Data providers may begin to use compression/chunking with confidence that most users and software can read it transparently, after relinking with netCDF-4
- Developers may adapt software to netCDF-4 format by relinking
- Developers may adapt software to enhanced data model incrementally, with examples that such adaptation is practical
- Upgrading software to make use of higher-level abstractions of netCDF-4 enhanced data model has significant benefits
  - Data providers can use more natural representation of complex data semantics
  - More natural conventions become possible
  - End users can access other types of data through netCDF APIs
- As we keep pushing common tasks into libraries, scientists can focus on doing science instead of data management

For more information

Web site: www.unidata.ucar.edu/netcdf/

Russ Rew: russ@unidata.ucar.edu

Extra Slides

New primitive types

- Unsigned numeric types better for representing data providers intent
  - ubyte: 8-bit unsigned interger
  - ushort: 16-bit unsigned integer
  - uint: 32-bit unsigned integer
- 64-bit integers needed for statistics and counts in large datasets
  - int64: 64-bit signed integer
  - uint64: 64-bit unsigned integer
- Variable-length strings an overdue improvement over character arrays
  - string: compact, variable-length strings

Groups

- Like directories in a file system, Groups provide name spaces and a hierarchy of containers
- Uses
  - Factoring out common information
    - Containers for data within regions, ensembles
    - Model metadata
  - Organizing a large number of variables
  - Providing name spaces for multiple uses of same names for dimensions, variables, attributes
  - Modeling large hierarchies

Variable-length types

Uses:

- Ragged arrays
- Modeling relational tables
- Nested with compound types for in situ observational data (profiles, soundings, time series)
- Example: observations along ocean tracks
  - each track has an ID, a description, and a variable-length list of profiles
    - each profile has a latitude, longitude, time, and a variable-length list of observations
      - each observation records pressure, temperature, and salinity at various depths

Compound types

Uses include:

- Representing vector quantities like wind
- Bundling multiple in situ observations together (profiles, soundings)
- Modeling relational database tuples
- Providing containers for related values of other user-defined types (strings, enums, ...)
- Representing C structures, Fortran derived types portably

Nested types

- Compound types may include other variable-length types or compound types as members
- Variable-length types may include other compound types or variable-length types as members
- Result is a potentially infinite number of user-defined data types
- Handling this in software can be new or intimidating to software developers

Guidance for developers

- Add support for netCDF enhanced data model features incrementally
  - new primitive types: unsigned numeric types and strings
  - nested Groups (simple recursion)
  - enumeration types (easy, no nesting)
  - opaque types (easy, no nesting)
  - compound types with only primitive members
  - compound types with fixed-size array members
  - variable-length arrays of primitives
  - compound types with members of user-defined type
  - variable-length arrays of user-defined types
- Look at nccopy for examples that read or write netCDF-4 data with all these features

Commitment to Compatibility

To ensure future access to existing data archives, Unidata is committed to compatibility of:

- **Data access:** new versions of netCDF software will provide read and write access to previously stored netCDF data.
- **Programming interfaces:** C and Fortran programs using documented netCDF interfaces from previous versions will work without change with new versions of netCDF software.
- **Future versions:** Unidata will continue to support both data access compatibility and program compatibility in future netCDF releases.